MODULE II (Neural Networks)

Introduction to neural networks - Single layer perceptrons, Multi Layer Perceptrons (MLPs), Representation Power of MLPs, Activation functions - Sigmoid, Tanh, ReLU, Softmax. Risk minimization, Loss function, Training MLPs with backpropagation, Practical issues in neural network training - The Problem of Overfitting, Vanishing and exploding gradient problems, Difficulties in convergence, Local and spurious Optima, Computational Challenges. Applications of neural networks.

Neural Networks

In Computer Science Neural Network means Artificial Neural Networks (ANN)

It is model for computation inspired from the learning process in human beings (or generally in biological organisms).

It is popular machine learning technique.

The human nervous system contains cells, which are referred to as neurons.

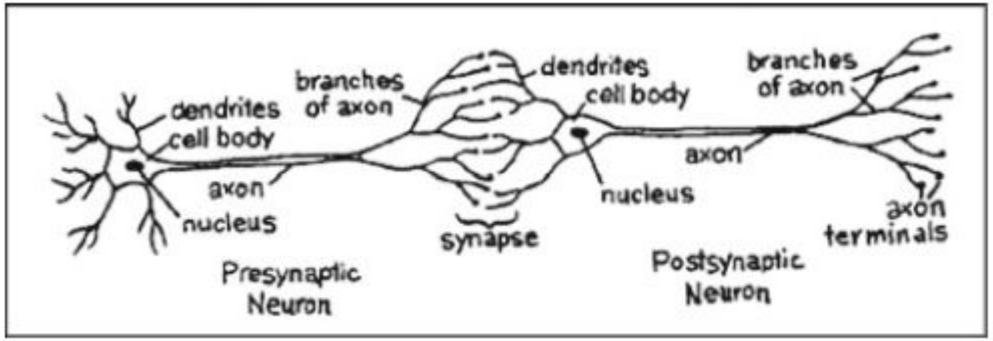
The neurons are connected to one another with the use of axons and dendrites, and the connecting regions between axons and dendrites are referred to as synapses.

The strengths of synaptic connections often change in response to external stimuli.

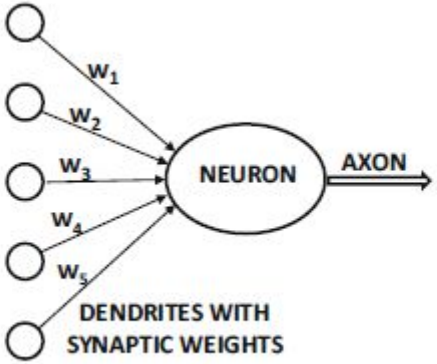
This change is how learning takes place in living organisms.

This biological mechanism is simulated in artificial neural networks, which contain computation units that are referred to as neurons.

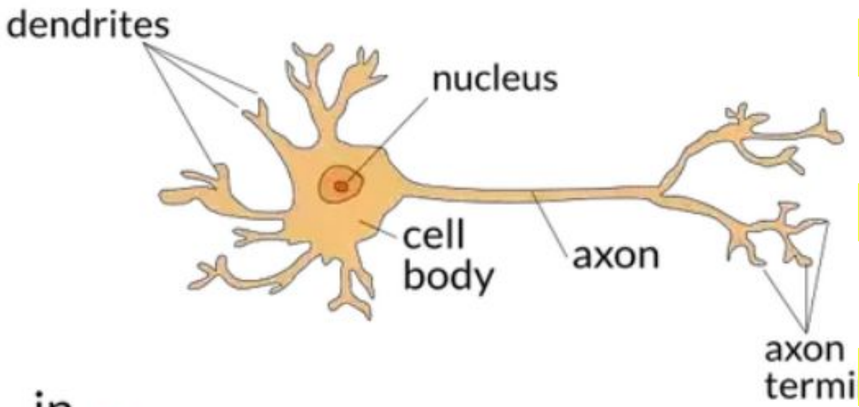
The computational units are connected to one another through weights, which serve the same role as the strengths of synaptic connections in biological organisms.

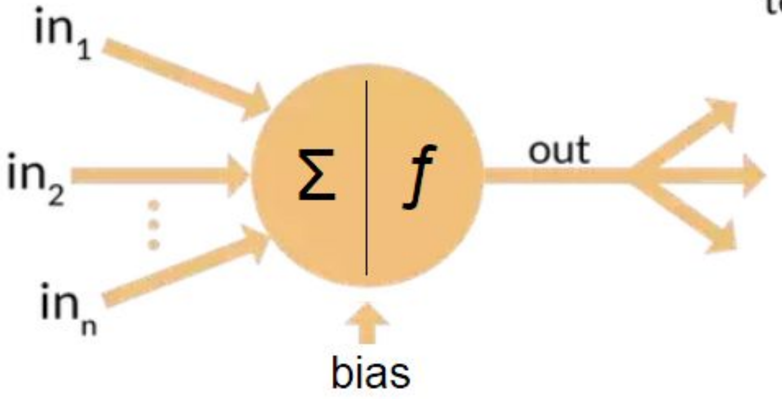


(a) Biological neural network



(b) Artificial neural network





dendrite: receives signals from other neurons

synapse: point of connection to other neurons

nucleus: processes the information

axon: transmits the output of this neuron

A single artificial neuron gets multiple inputs or signals, (say $x_1, x_2, x_3, \ldots$)

Input signals are attenuated(multiplied) by the weights along the edges. (say $w_1, w_2, w_3, \ldots$)

Bias (b) is a fixed value given to the neuron

The sum of incoming signals to a neuron is computed as,

Net input = $b+\sum x_i w_i$

Output of a neuron, or response is computed through activation function

Ie., Response,or output or, y = f(Net input), where f is the activation function.

Learning in Neural Networks

An artificial neural network computes a function of the inputs by propagating the computed values from the input neurons to the output neuron(s) and using the weights as intermediate parameters.

Learning occurs by changing the weights connecting the neurons.

Just as external stimuli are needed for learning in biological organisms,the external stimulus in artificial neural networks is provided by the training data containing examples of input-output pairs of the function to be learned.

For example, the training data might contain pixel representations of images (input) and their annotated labels (e.g., carrot, banana) as the output.

These training data pairs are fed into the neural network by using the input representations to make predictions about the output labels.

The training data provides feedback to the correctness of the weights in the neural network depending on how well the predicted output (e.g., probability of carrot) for a particular input matches the annotated output label in the training data

One can view the errors made by the neural network in the computation of a function as a kind of unpleasant feedback in a biological organism, leading to an adjustment in the synaptic strengths.

Similarly, the weights between neurons are adjusted in a neural network in response to prediction errors.

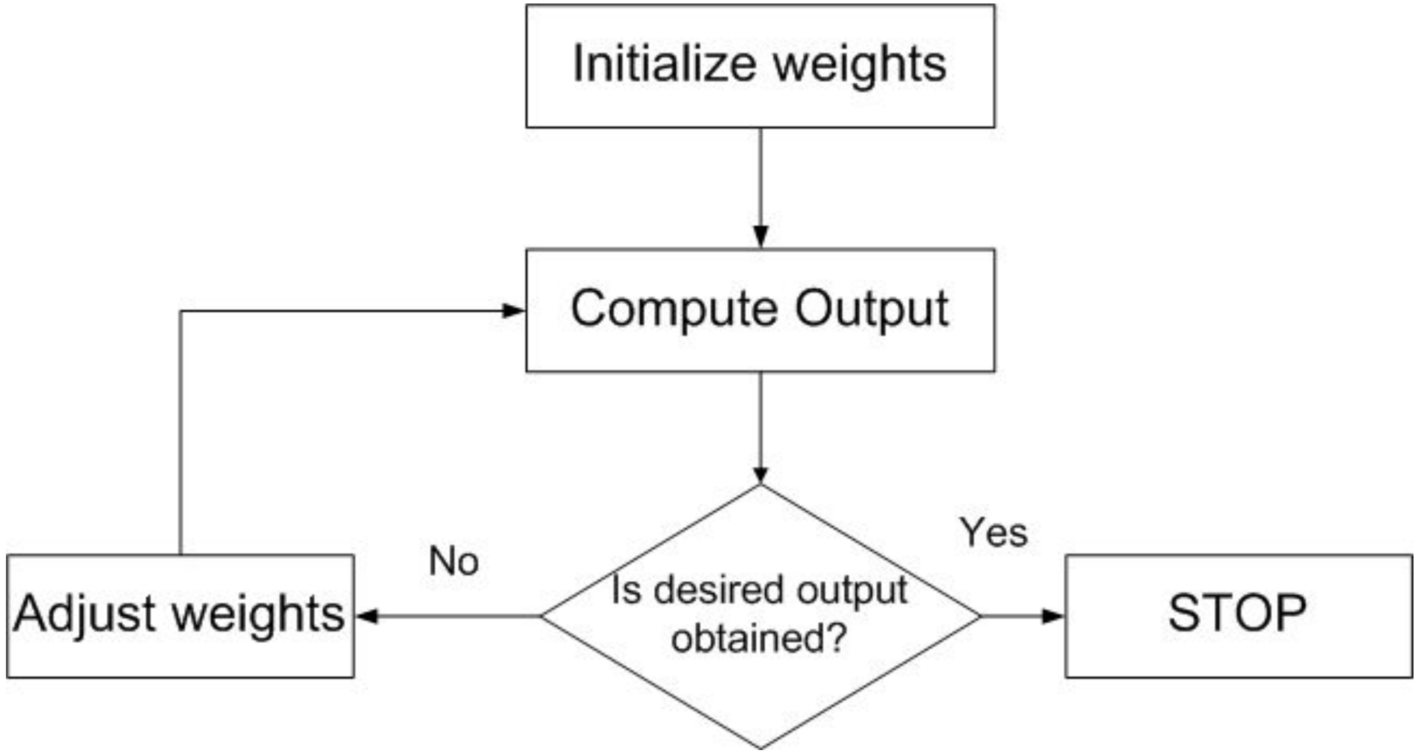
The goal of changing the weights is to modify the computed function to make the predictions more correct in future iterations.

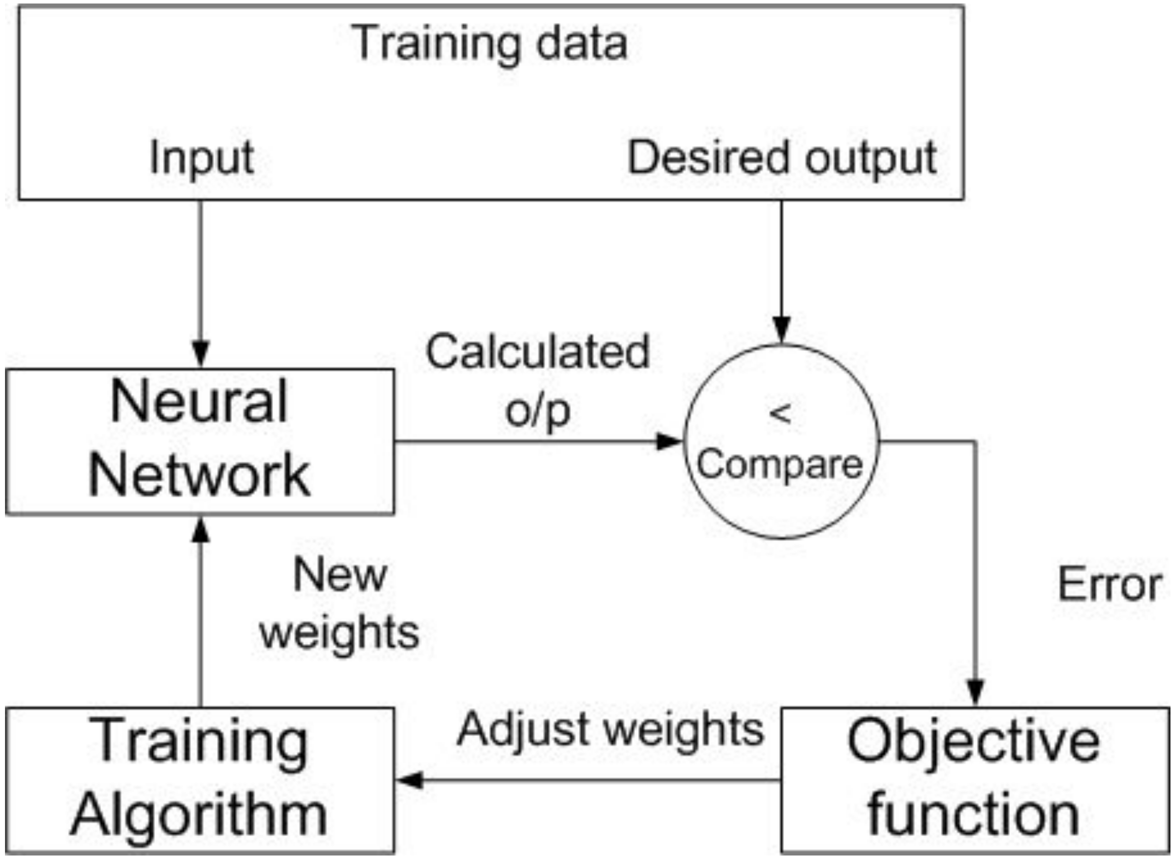
Therefore, the weights are changed carefully in a mathematically justified way so as to reduce the error in computation on that example.

By successively adjusting the weights between neurons over many input-output pairs, the function computed by the neural network is refined over time so that it provides more accurate predictions.

Therefore, if the neural network is trained with many different images of bananas, it will eventually be able to properly recognize a banana in an image it has not seen before.

This ability to accurately compute functions of unseen inputs by training over a finite set of input-output pairs is referred to as model generalization.

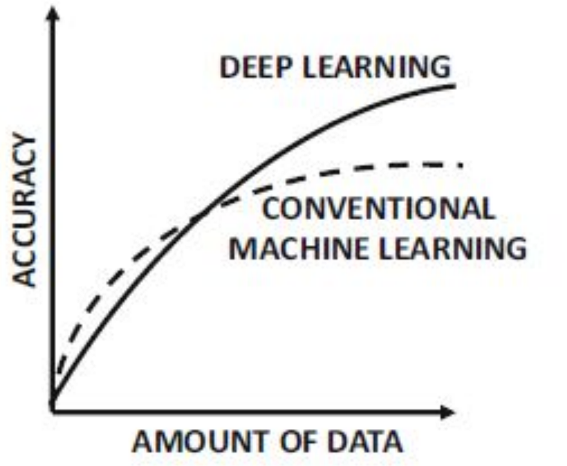

Initialize weights
Compute Output
Is desired output obtained?
No
Yes
Adjust weights
STOP


Training data
Input
Desired output
Neural Network
Calculated o/p
<
Compare
Error
New weights
Training Algorithm
Adjust weights
Objective function

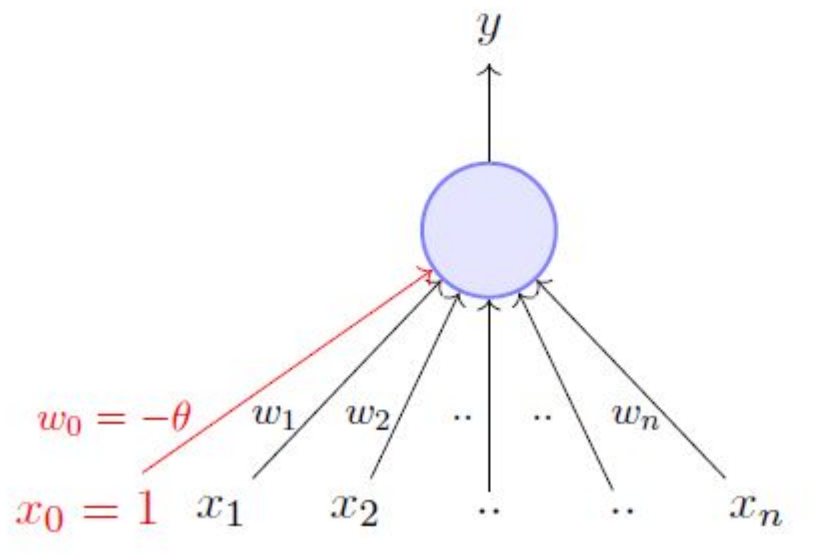
Deep learning is a kind of learning using deep neural network.

‘deep’ neural network is a neural network that has more than 3 layers of neurons.

In recent years deep learning has shown an accuracy on some tasks that exceeds that of a human.



Single Layer Perceptron



Single Layer Perceptron

$x_1, x_2, \ldots\ldots, x_n$ - Inputs
$w_1, w_2, \ldots\ldots, w_n$ - Weights
y - output -> binary

$-\theta$ = bias, bias is written as $w_0 * x_0$

Numerical weights for inputs can be assigned and a mechanism for learning these weights is present.

Inputs can be real numbers or boolean values.

Decision making in single layer perceptron

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

Consider two vectors **w** and **x**

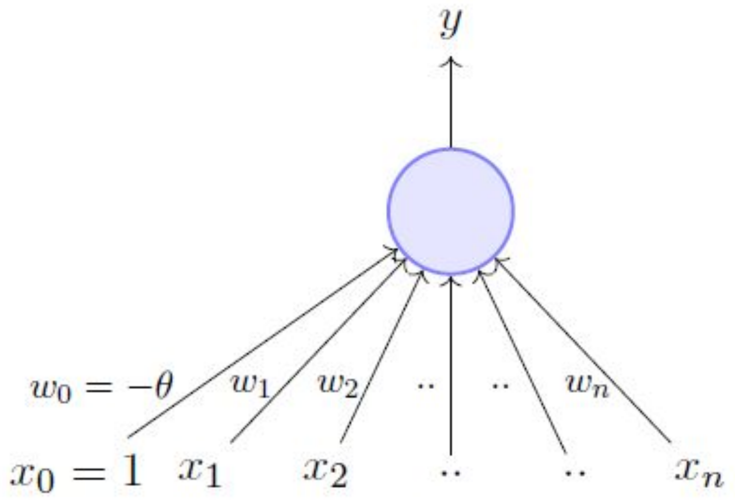
$$\mathbf{w} = [w_0, w_1, w_2, ..., w_n]$$
$$\mathbf{x} = [1, x_1, x_2, ..., x_n]$$
$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^{\mathbf{T}}\mathbf{x} = \sum_{i=0}^{n} w_i * x_i$$

We can thus rewrite the perceptron rule as

$$y = 1 \quad if \quad \mathbf{w}^{\mathbf{T}}\mathbf{x} \geq 0$$
$$= 0 \quad if \quad \mathbf{w}^{\mathbf{T}}\mathbf{x} < 0$$

Algorithm: Perceptron Learning Algorithm

```
$P \leftarrow inputs \quad with \quad label \quad 1$;
$N \leftarrow inputs \quad with \quad label \quad 0$;
Initialize $\mathbf{w}$ randomly;
while $!convergence$ do
    Pick random $\mathbf{x} \in P \cup N$ ;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ then
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$ ;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ then
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$ ;
    end
end
//the algorithm converges when all the
 inputs are classified correctly
```



$x_1 = isActorDamon$

$x_2 = isGenreThriller$

$x_3 = isDirectorNolan$

$x_4 = imdbRating(scaled\ to\ 0\ to\ 1)$

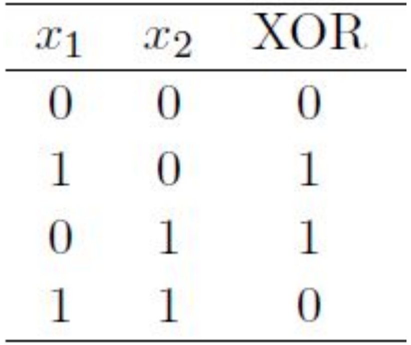
... ...

$x_n = criticsRating(scaled\ to\ 0\ to\ 1)$

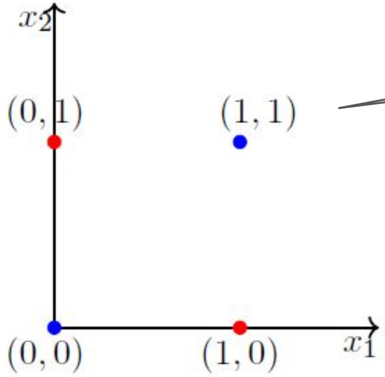
- Consider the problem of deciding whether to watch a movie or not
- Suppose we are given a list of m movies and a label (class) associated with each movie indicating whether the user liked this movie or not : binary decision
- Further, suppose we represent each movie with n features (some boolean, some real valued)
- The perceptron can learn the weights so that it can predict whether one is going to watch this movie or not based on the features.

A single perceptron cannot be used to model data which is not linearly separable.

Consider the XOR boolean function.



x_1	x_2	XOR
0	0	0
1	0	1
0	1	1
1	1	0



Linearly NOT separable

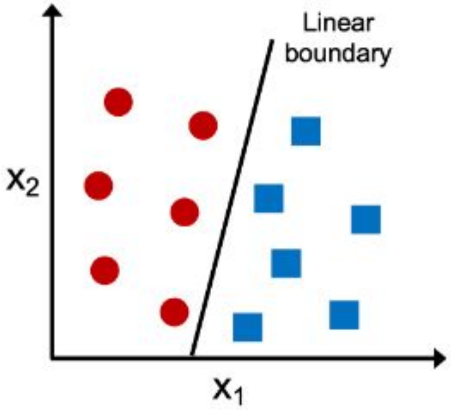
It is impossible to draw a line which separates the red points from the blue points.

Such data is said to be linearly not separable.

A network of perceptrons can deal with data which is not linearly separable.

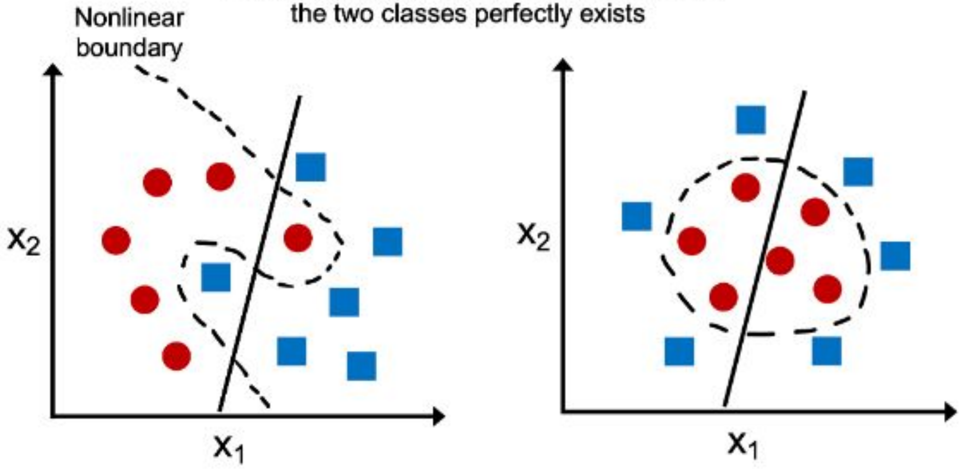
Linearly separable

A linear decision boundary that separates the two classes exists



Not linearly separable

No linear decision boundary that separates the two classes perfectly exists



Multilayer Perceptrons (MLP)

Networks containing, an input, output and one or more hidden layers are called Multilayer Perceptrons (MLP, in short)

In multi-layer neural networks, the neurons are arranged in layered fashion, in which the input and output layers are separated by a group of hidden layers.

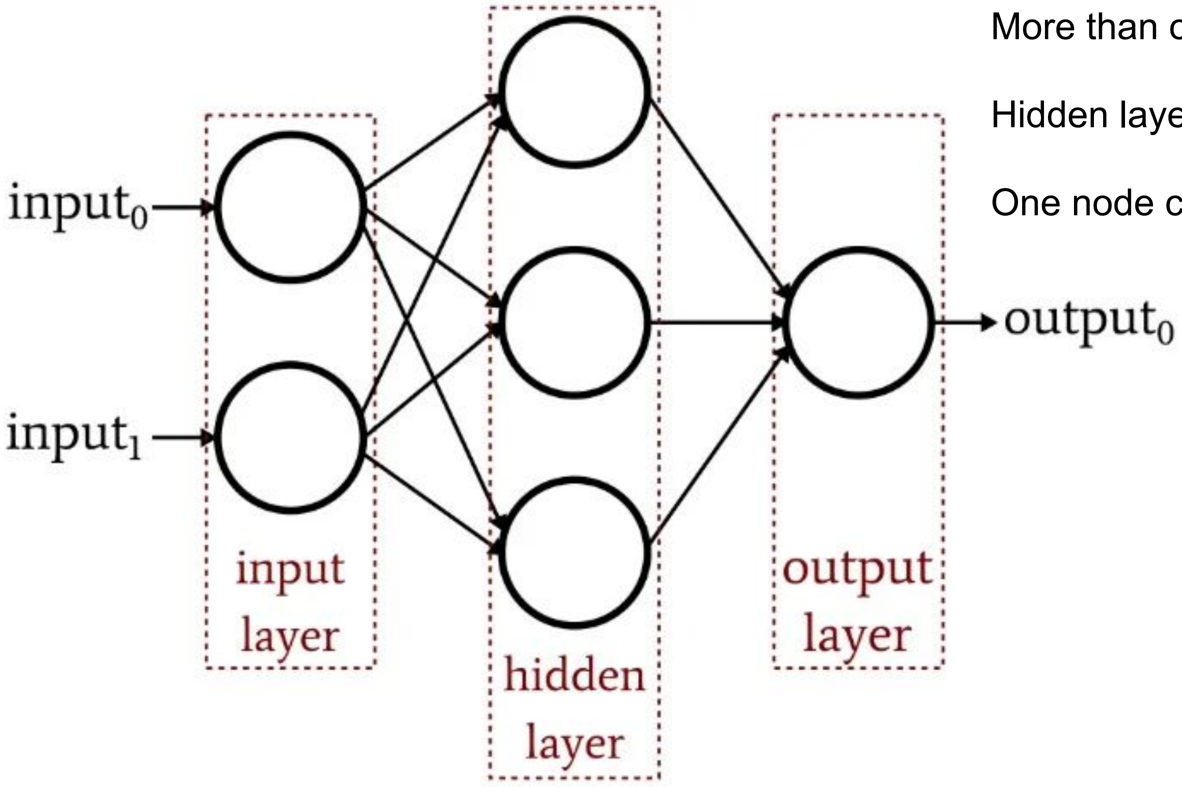
This layer-wise architecture of the neural network is also referred to as a feed-forward network

Can have more than one input and

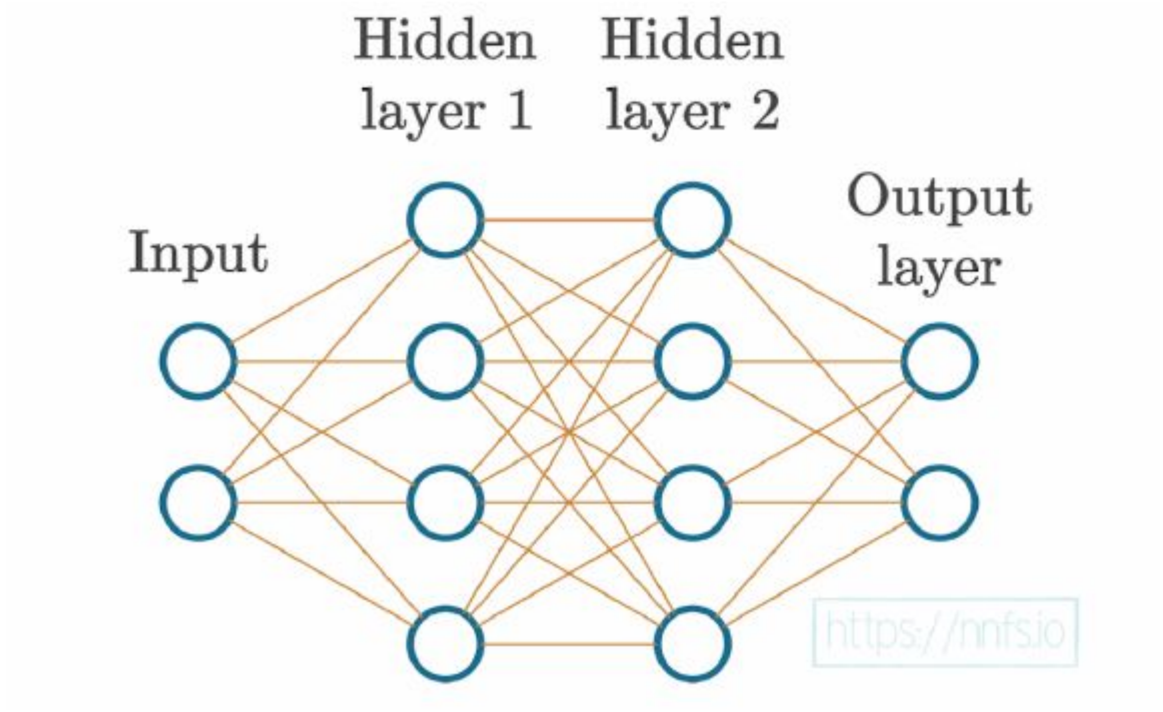
More than one output

Hidden layer can contain any no. of nodes

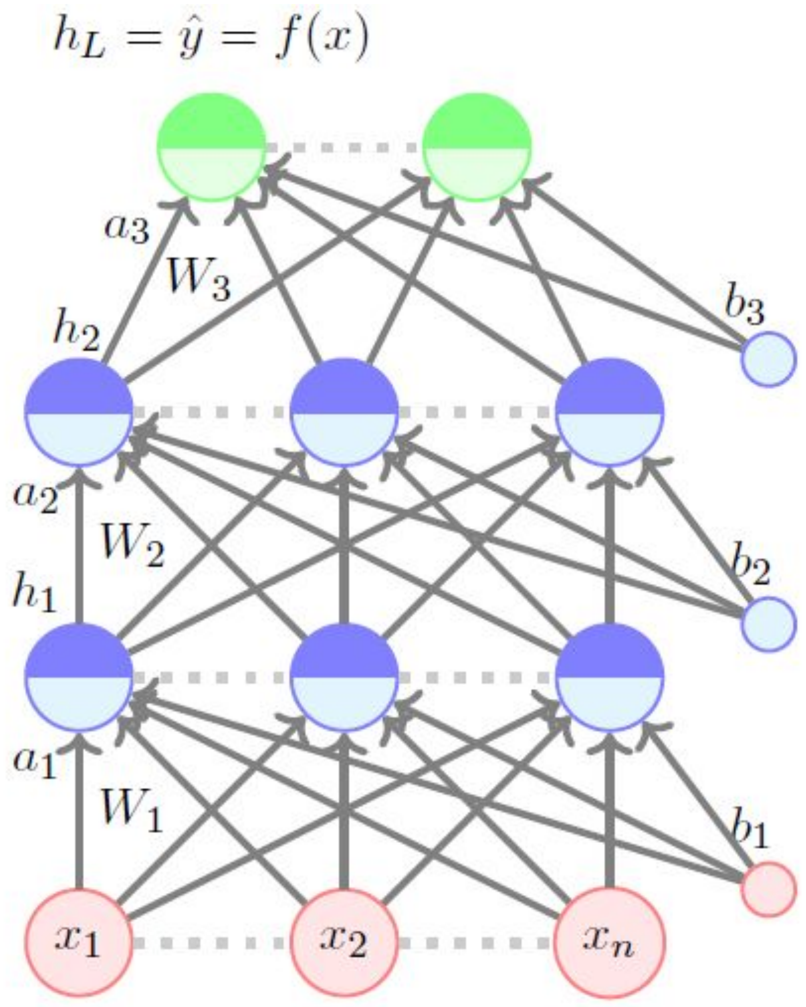
One node corresponds to one neuron



A feed forward neural network with 3 layers



A feed forward neural network with 4 layers



Feed forward neural network with 2 hidden layers

Activation Function

In the perceptron model, activation function is binary step function.

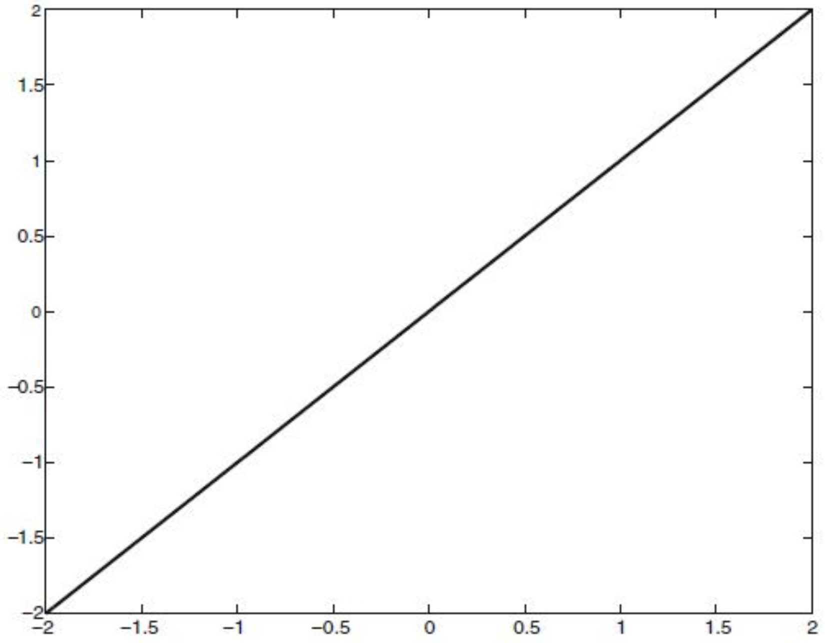
Ie, output = 1, if, sum of inputs>= 0 and output = 0, if, sum of inputs <0

Sum of inputs to a neuron can fed into any mathematical function called activation function.

The following describes commonly used activation functions in ANN.

a) Identity Function (Linear)

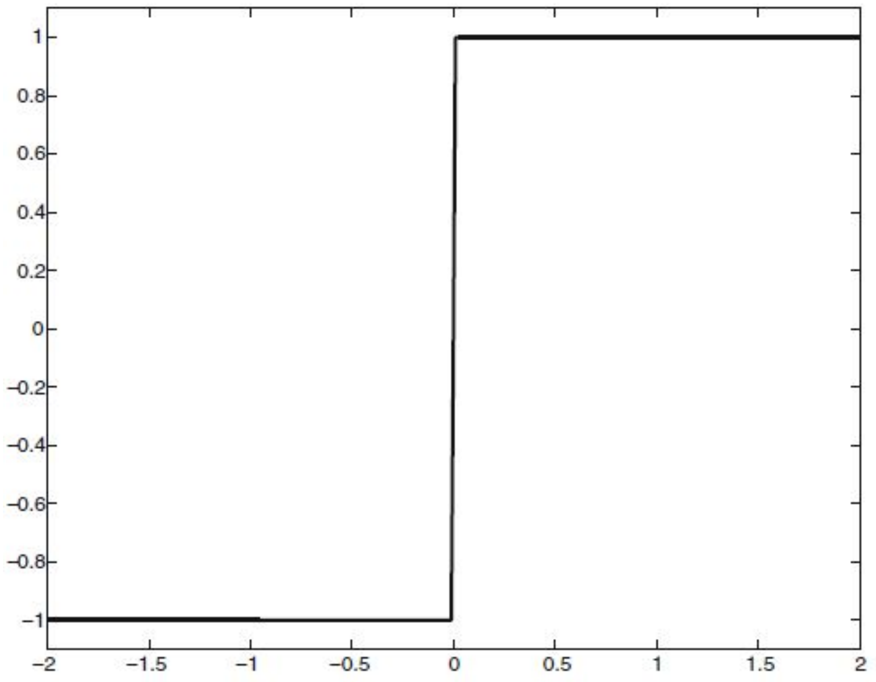
$f(x) = x$



(a) Identity

b) Sign Function

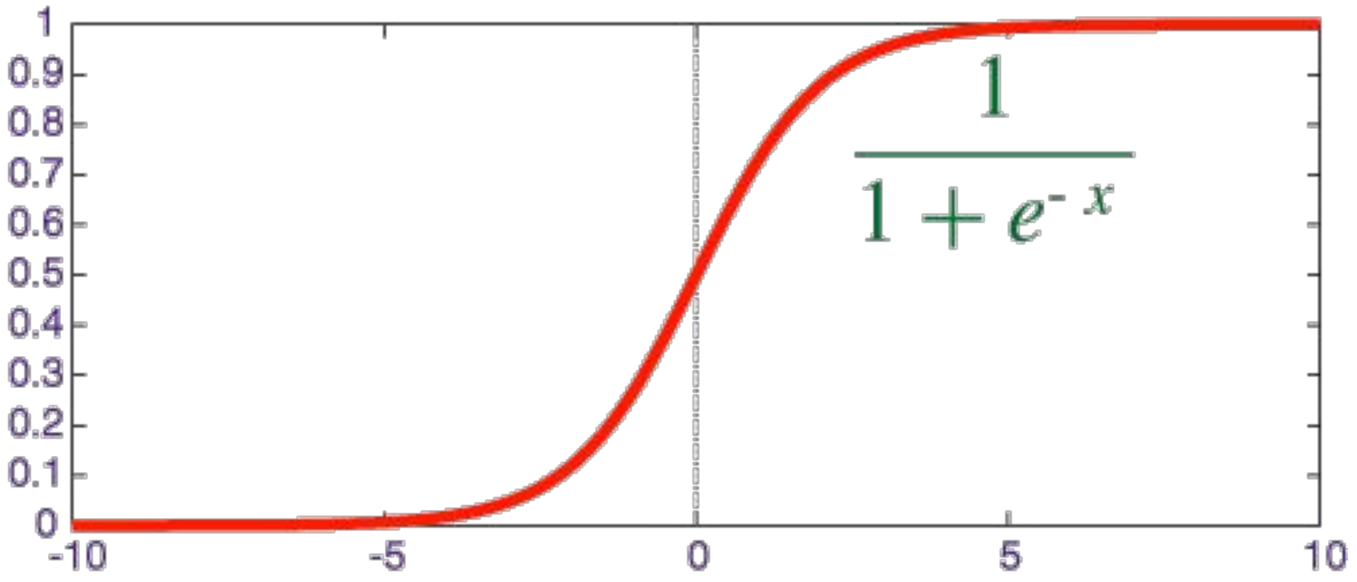
$$f(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$



(b) Sign

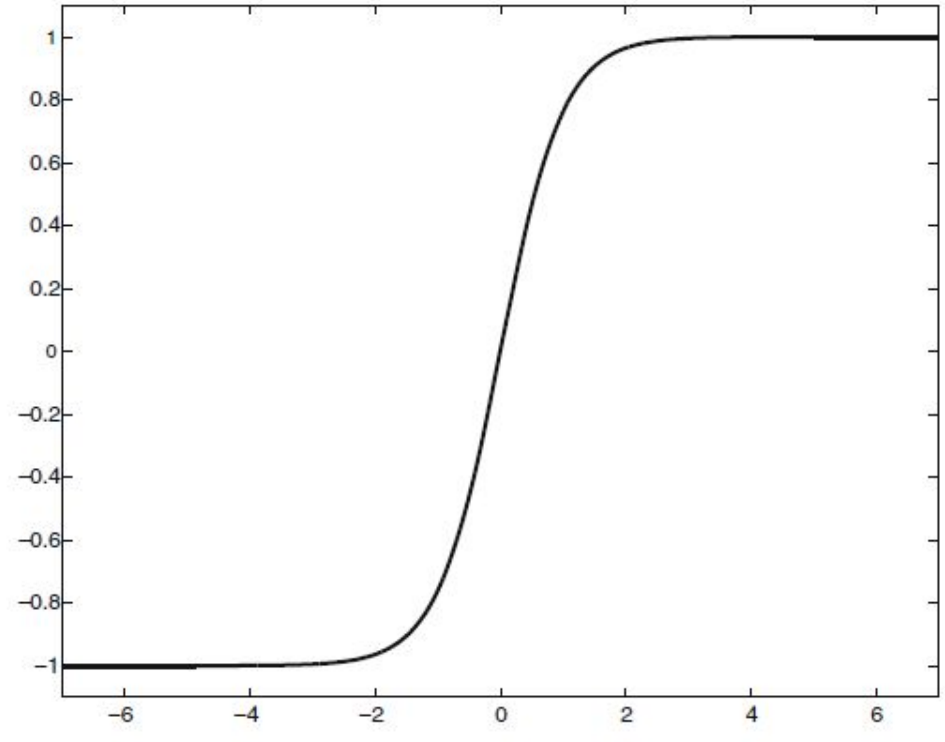
c) Sigmoid Function

$$f(x) = \frac{1}{1+e^{-(x)}}$$



d) tanh function (Hyperbolic tangent)

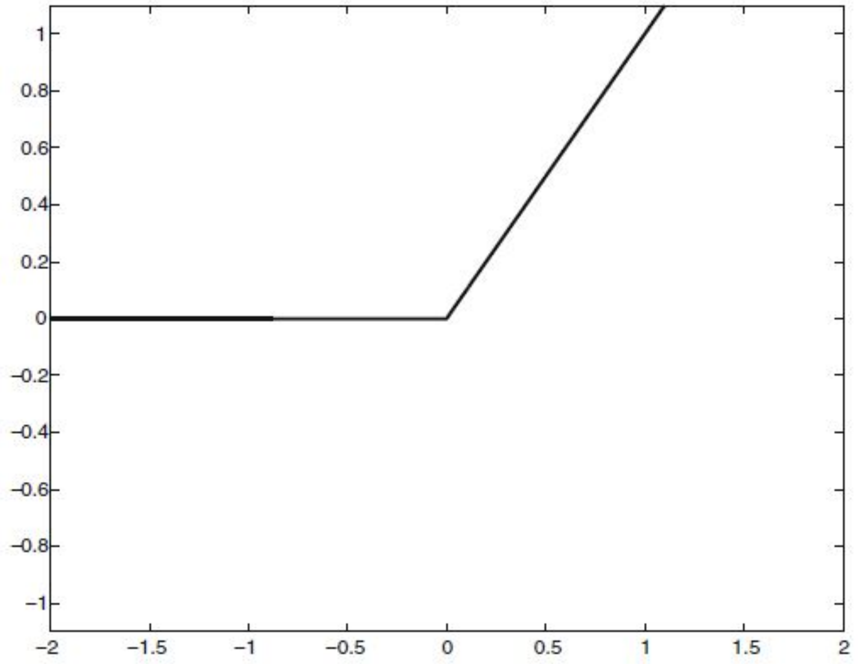
$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$



(d) Tanh

e) ReLU Function (Rectified Linear Unit)

$f(x) = max(0,x)$



(e) ReLU

e) softmax function

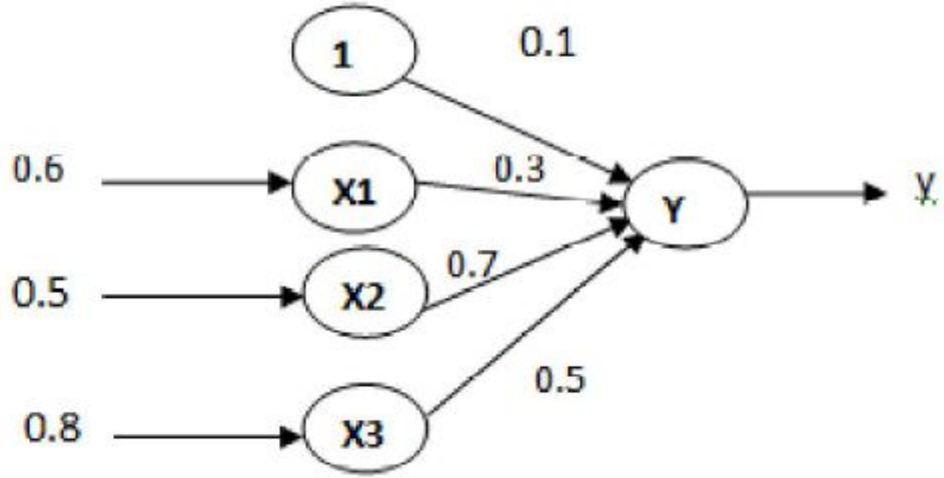
Applied to a vector.

All entries in the softmax output vector are between 0 and 1.(probability values)

Used in multi class classification

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Calculate the output of the following neuron Y with the activation function as a) binary sigmoid b) tanh c)ReLU



Representation Power of MLPs

A MLP with a single hidden layer can represent any boolean function.

Any boolean function of n inputs can be represented exactly by a network of perceptrons containing 1 hidden layer with 2^n perceptrons and one output layer containing 1 perceptron.

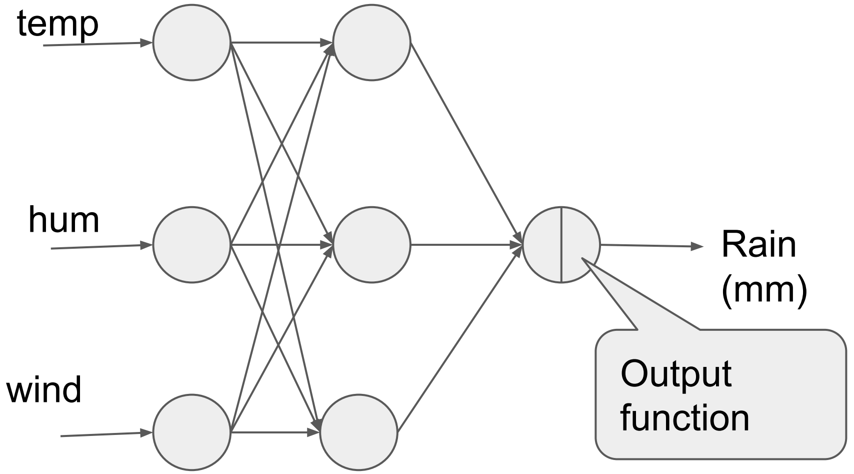
Representation Power of a Multilayer Network of Sigmoid Neurons

A multilayer network of neurons with a single hidden layer can be used to approximate any continuous function to any desired precision.

Output function

The activation function in the output layer is called output unction.

If the output value required is Real (regression problem), then output function will be the linear function.



Output function = linear function

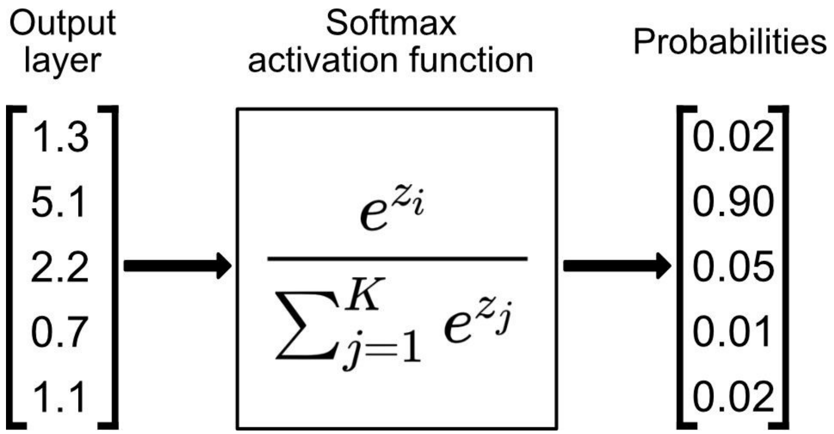
ie, $y = f(x) = x$

Activation function at input & hidden layers remain as sigmoid.

Output function

For classification problems, output required is the probability distribution of class labels and so most suitable output function will be softmax function.

Real values in output layer are transformed into probabilities.


Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

Output function

Calculate the output of softmax function for the following vector.

$$\begin{bmatrix} -1 \\ 0 \\ 3 \\ 5 \end{bmatrix}$$

Loss Function/Error Function/Objective Function

Used to evaluate how well a machine learning algorithm has performed.

a) Mean Square Error / Quadratic Loss / L2 Loss

Mean square error (MSE) is measured as the average of squared difference between predictions and actual observations.

It is a loss function used in regression problems.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

, where, n is the number of outputs, Y_i is the actual (target output and $\hat{Y}_i$ is the predicted output.

b) Mean Absolute Error / L1 Loss

Average of absolute differences between the actual and the predicted value
Used in regression.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right|$$

Calculate the MSE and MAE in the following case.

Actual output Predicted

$$\begin{bmatrix} 5 \\ 2 \\ 1 \\ 3 \end{bmatrix} \qquad \begin{bmatrix} 4.7 \\ 2.2 \\ 0.9 \\ 3.4 \end{bmatrix}$$

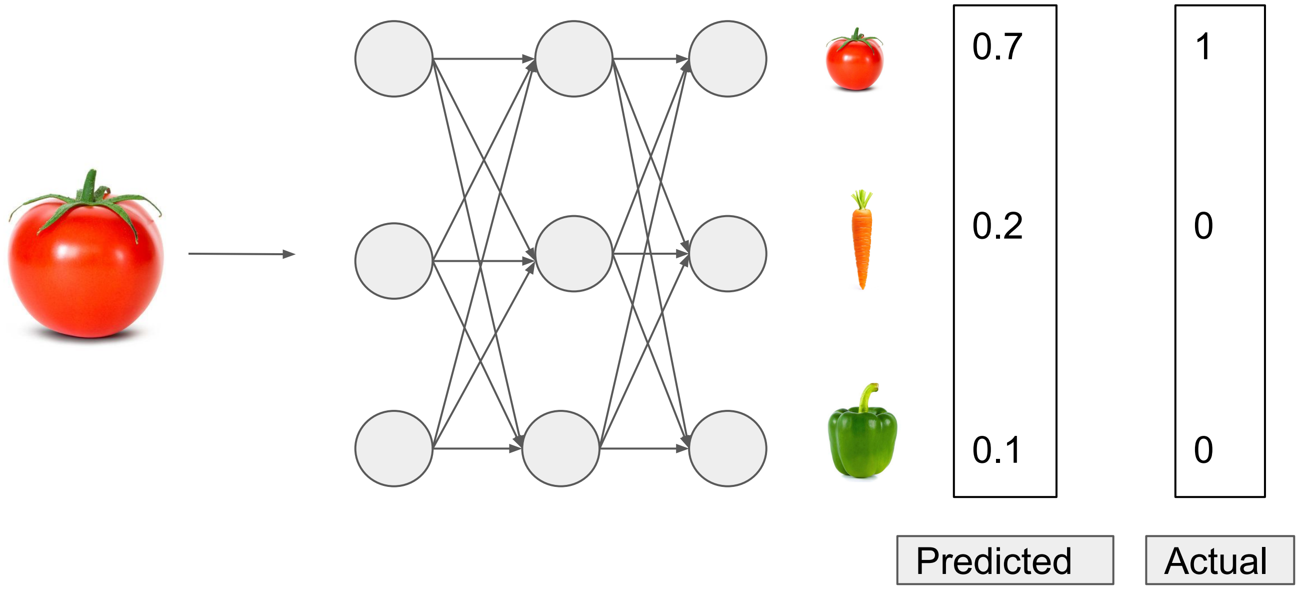
c) Cross Entropy

This is used to capture the difference between two probability distributions.

In classification problems, output is a probability distribution.

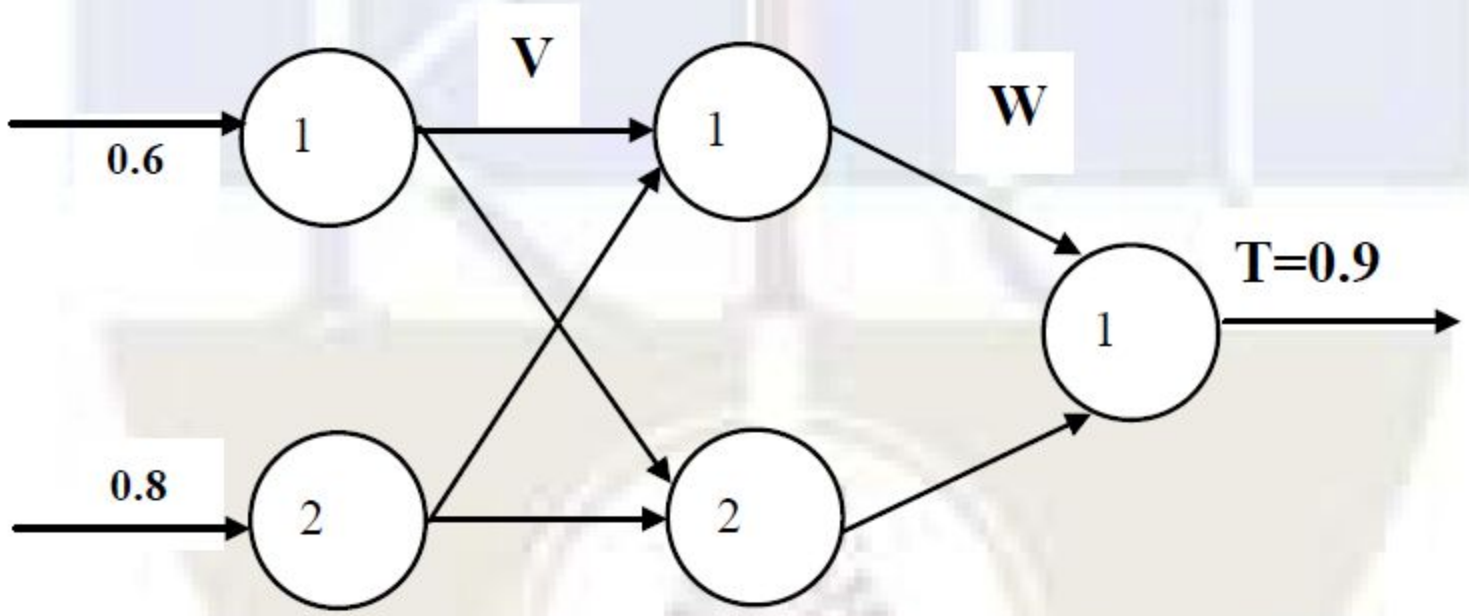
So cross entropy is used to calculate the difference between actual and expected classes.

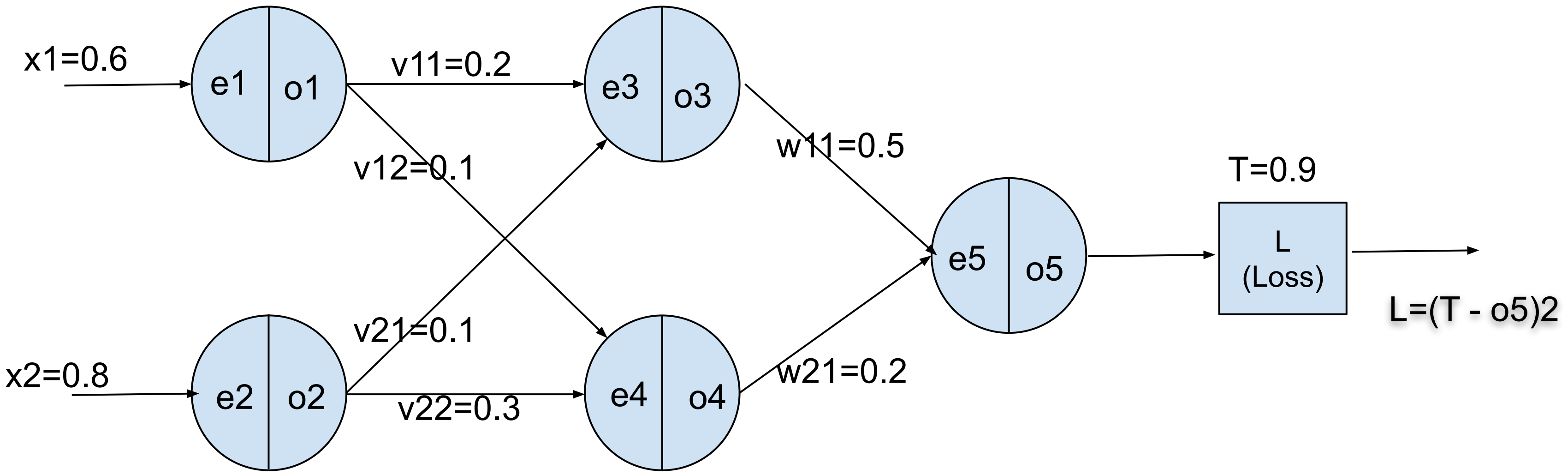
Cross entropy = $-\sum_{c=1}^{k} y_c \log \hat{y}_c$ where, k is the no. of classes, yc is the actual class probability and y^ is the predicted class probability.



Cross entropy = -(1*log 0.7 + 0*log 0.2 + 0*log 0.1) =

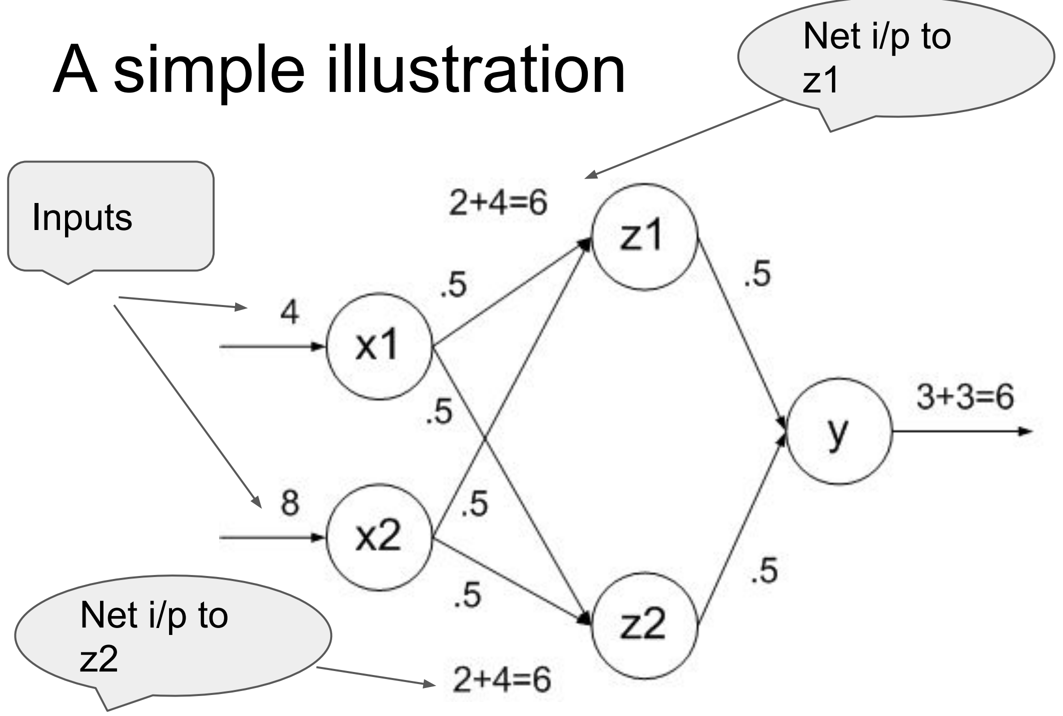
Update the parameters V_{11} in the given MLP using back propagation with learning rate as 0.5 and activation function as sigmoid. Initial weights are given as $V_{11}= 0.2$, $V_{12}=0.1$, $V_{21}=0.1$, $V_{22}=0.3$, $V_{11}=0.2$, $W_{11}=0.5$, $W_{21}=0.2$





Refer to the problem worked out in class.

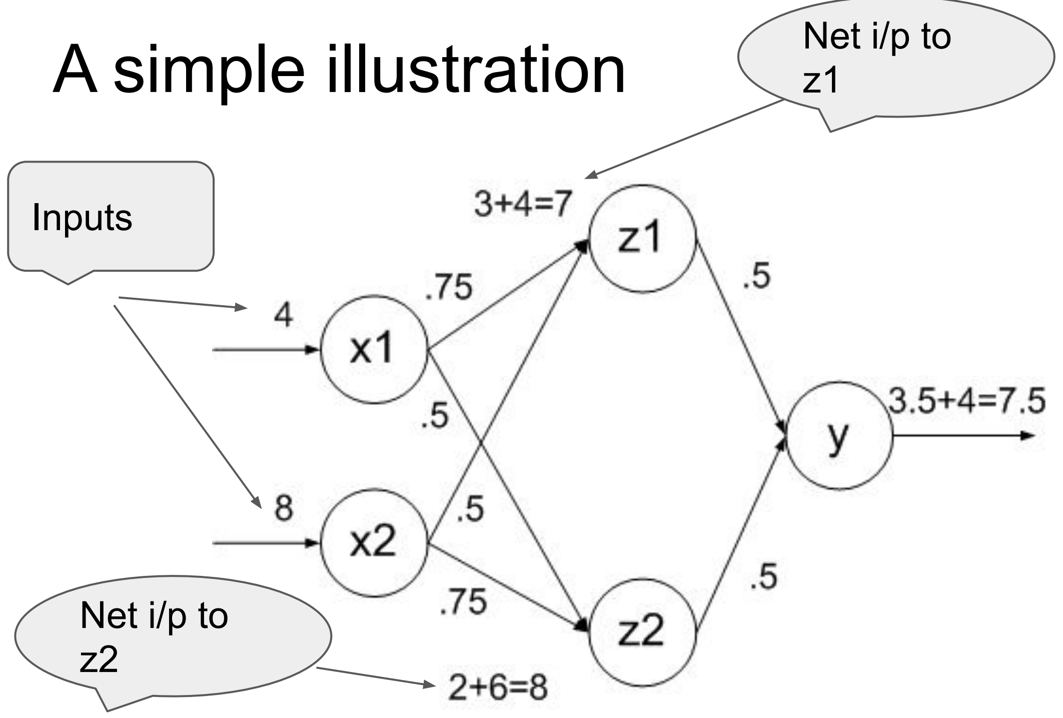
A simple illustration



Error= 10-6=4, this error is back propagated and a new weight is computed for all edges. Let the new network be as follows

This BPNN has 2 input nodes(x1 and x2), 2 hidden layer nodes(z1 and z2), and one output node (y).
Let the network is to be trained for (4,8) -> 10,
Assume, the initial weights as indicated above and assume that all bias = 0 and activation function is identity function. (so, output of a neuron is same as its input)

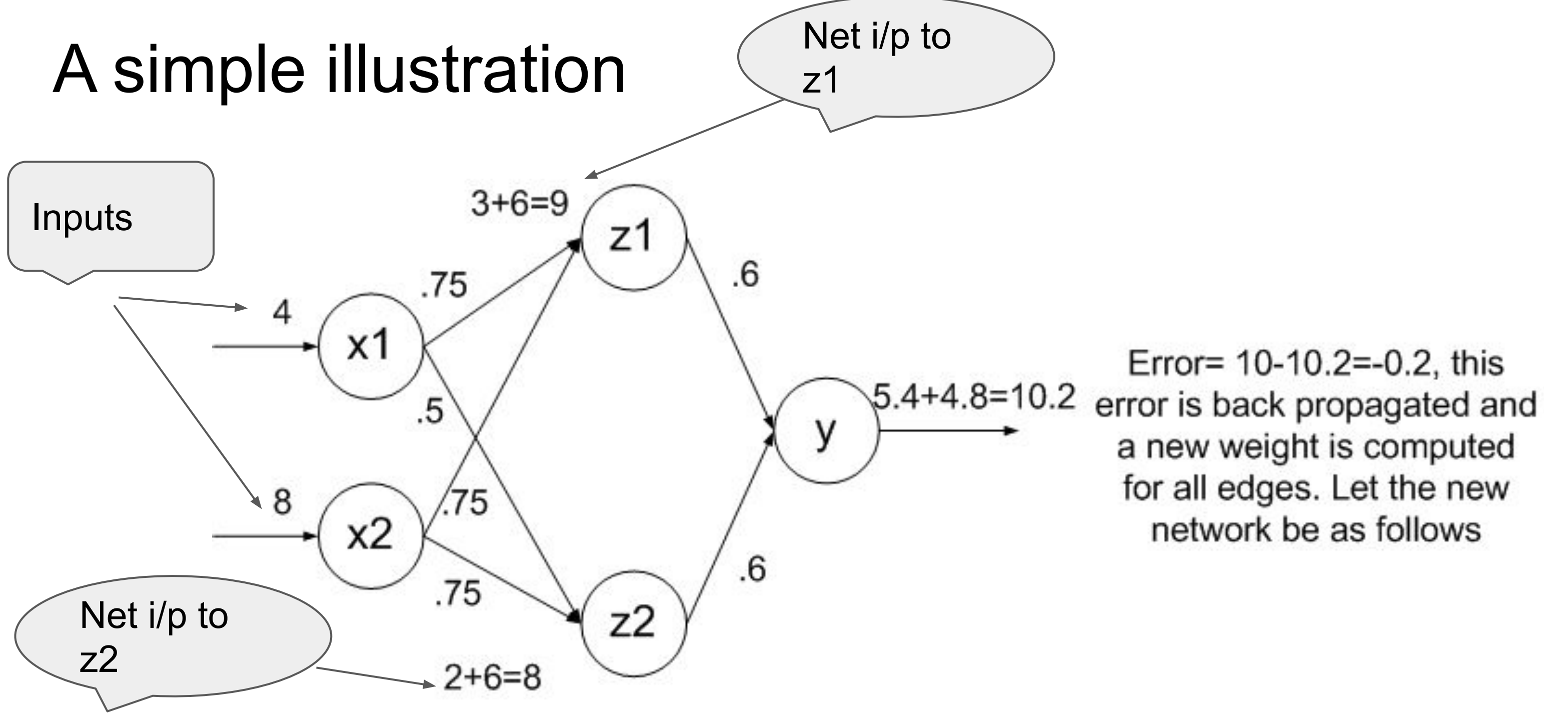
A simple illustration



Error= 10-7.5=2.5, this error is back propagated and a new weight is computed for all edges. Let the new network be as follows

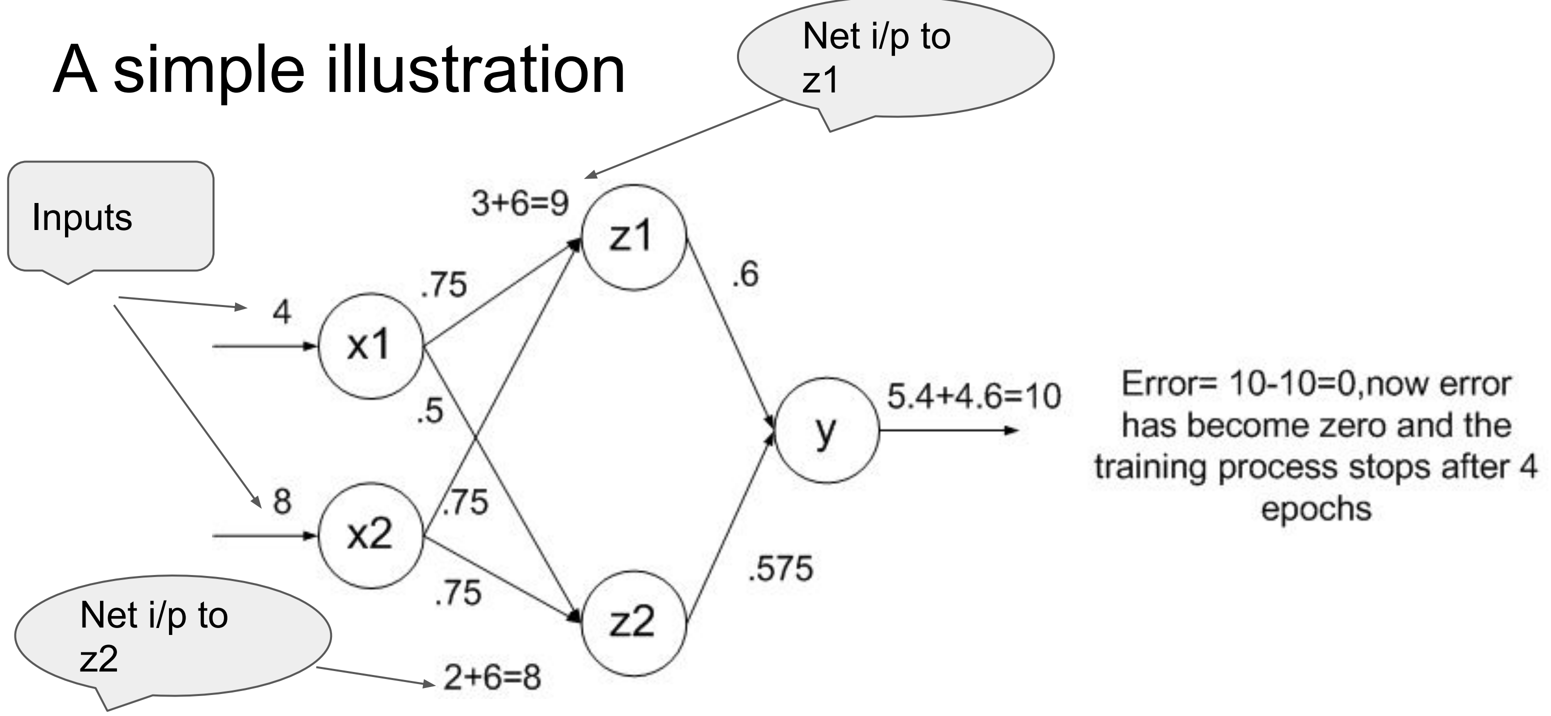
This BPNN has 2 input nodes(x1 and x2), 2 hidden layer nodes(z1 and z2), and one output node (y).
Let the network is to be trained for (4,8) -> 10,
Assume, the initial weights as indicated above and assume that all bias = 0 and activation function is identity function. (so, output of a neuron is same as its input)

A simple illustration


Net i/p to z1
Inputs
3+6=9
z1
.75
.6
4
x1
.5
y
5.4+4.8=10.2
Error= 10-10.2=-0.2, this error is back propagated and a new weight is computed for all edges. Let the new network be as follows
8
.75
x2
.6
.75
z2
Net i/p to z2
2+6=8

This BPNN has 2 input nodes(x1 and x2), 2 hidden layer nodes(z1 and z2), and one output node (y).
Let the network is to be trained for (4,8) -> 10,
Assume, the initial weights as indicated above and assume that all bias = 0 and activation function is identity function. (so, output of a neuron is same as its input)

A simple illustration


Net i/p to z1
Inputs
3+6=9
z1
.75
.6
4
x1
5.4+4.6=10
.5
y
Error= 10-10=0,now error has become zero and the training process stops after 4 epochs
8
.75
x2
.575
.75
z2
Net i/p to z2
2+6=8

This BPNN has 2 input nodes(x1 and x2), 2 hidden layer nodes(z1 and z2), and one output node (y).
Let the network is to be trained for (4,8) -> 10,
Assume, the initial weights as indicated above and assume that all bias = 0 and activation function is identity function. (so, output of a neuron is same as its input)

Implementations

NN regression

i/p	o/p
(2,3)	6
(2,4)	8
(2,5)	10
(2,6)	12
(2,7)	14
(2,8)	16
(2,30)	?
(2,40)	?

$(x_1, x_2) \rightarrow y$

i/p array

$[[2,3],[2,4],[2,5],[2,6],[2,7],[2,8]] = X$

o/p array

$[6, 8, 10, 12, 14, 16] - y$

1. Create a regressor

```
reg = MLPRegressor(hidden_layer_sizes=(2,2),random_state=1,
max_iter=5000)
```

2. Train the regressor

$$\text{reg.fit}(\underset{\uparrow \text{ i/p}}{X}, \underset{\text{o/p}}{y})$$

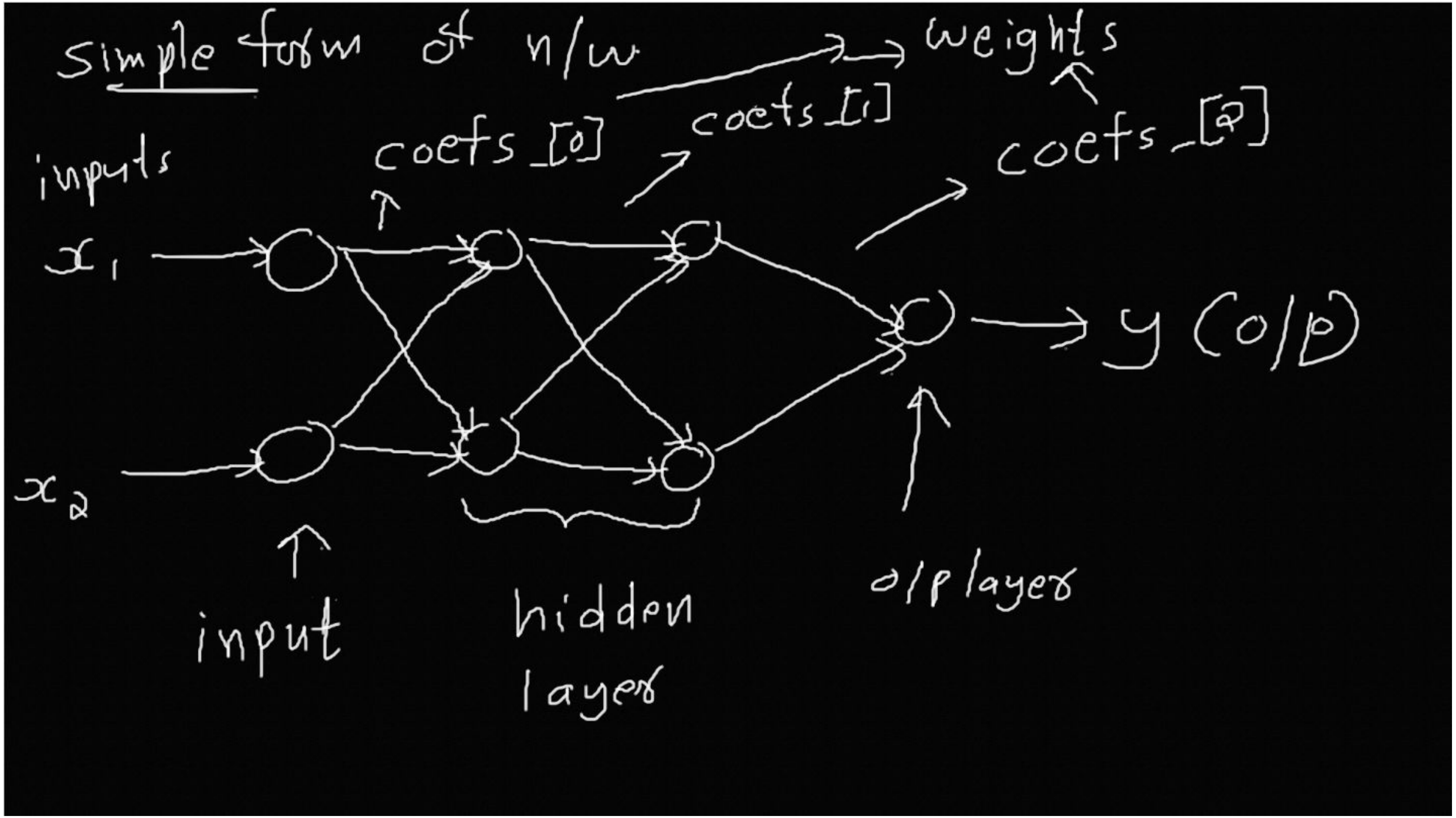
3. Predict o/p for new inputs.

```
reg.predict([[2,30],[2,40]])
```

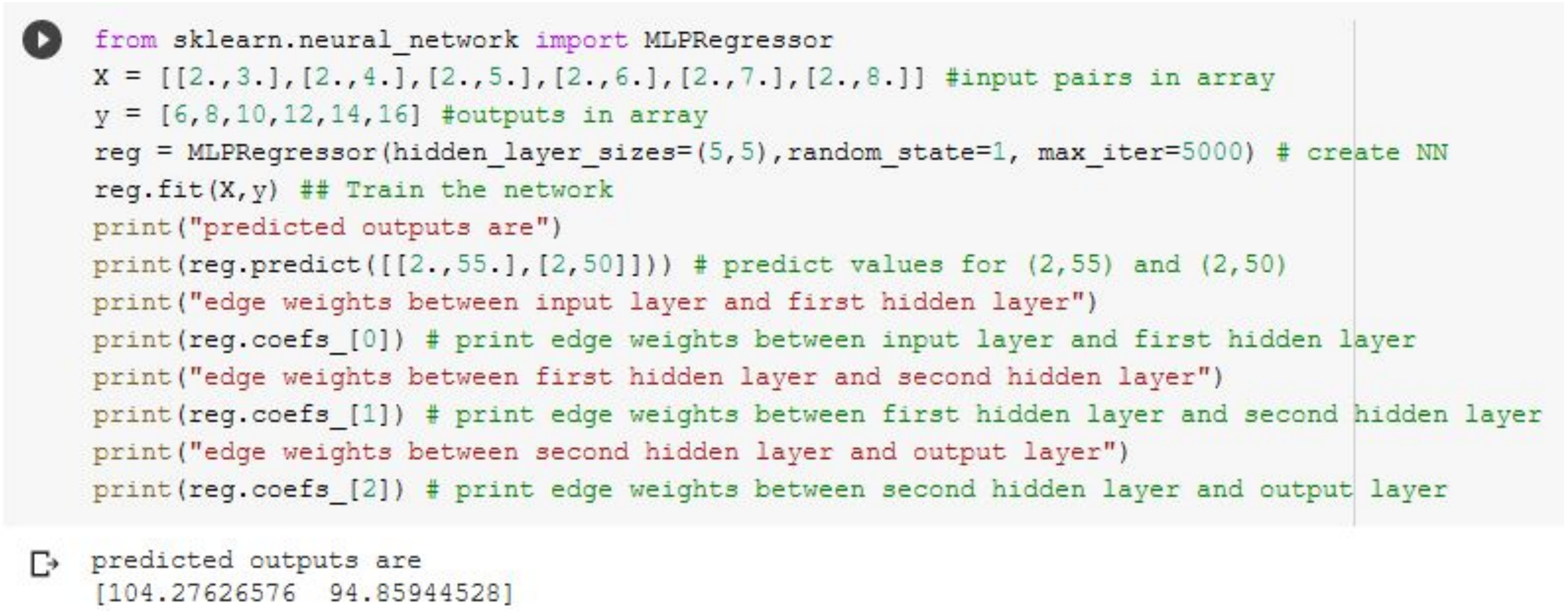
$$\text{O/p will be displayed as } \underbrace{\text{an}}_{\text{second o/p}} \text{ array of two values}$$

displayed → one o/p

an → second o/p


simple form of n/w
weights
coefs_[0]
coefs_[1]
coefs_[2]
inputs
x_1
x_2
y (o/p)
input
hidden layer
o/p layer

Neural Network Regressor



```
from sklearn.neural_network import MLPRegressor
X = [[2.,3.],[2.,4.],[2.,5.],[2.,6.],[2.,7.],[2.,8.]] #input pairs in array
y = [6,8,10,12,14,16] #outputs in array
reg = MLPRegressor(hidden_layer_sizes=(5,5),random_state=1, max_iter=5000) # create NN
reg.fit(X,y) ## Train the network
print("predicted outputs are")
print(reg.predict([[2.,55.],[2,50]])) # predict values for (2,55) and (2,50)
print("edge weights between input layer and first hidden layer")
print(reg.coefs_[0]) # print edge weights between input layer and first hidden layer
print("edge weights between first hidden layer and second hidden layer")
print(reg.coefs_[1]) # print edge weights between first hidden layer and second hidden layer
print("edge weights between second hidden layer and output layer")
print(reg.coefs_[2]) # print edge weights between second hidden layer and output layer
```

```
predicted outputs are
[104.27626576  94.85944528]
```

Neural Network Classifier

i/p vectors

o/p class

[[100, 100, 220, 50, 60], — 1 Tomato

[50, 300, 180, 90, 100], — 2 — Carrot

[75, 900, 50, 220, 85]] — 3 — Capsicum

$\uparrow$

X

[1, 2, 3]

$\uparrow$

y

It can be given for training

1. Create a NN classifier,

```
clf = MLPClassifier(hidden_layer_sizes=(10, 5), random_state=1,max_iter=500)
```

↑
classifier object

2. Train the classifier using

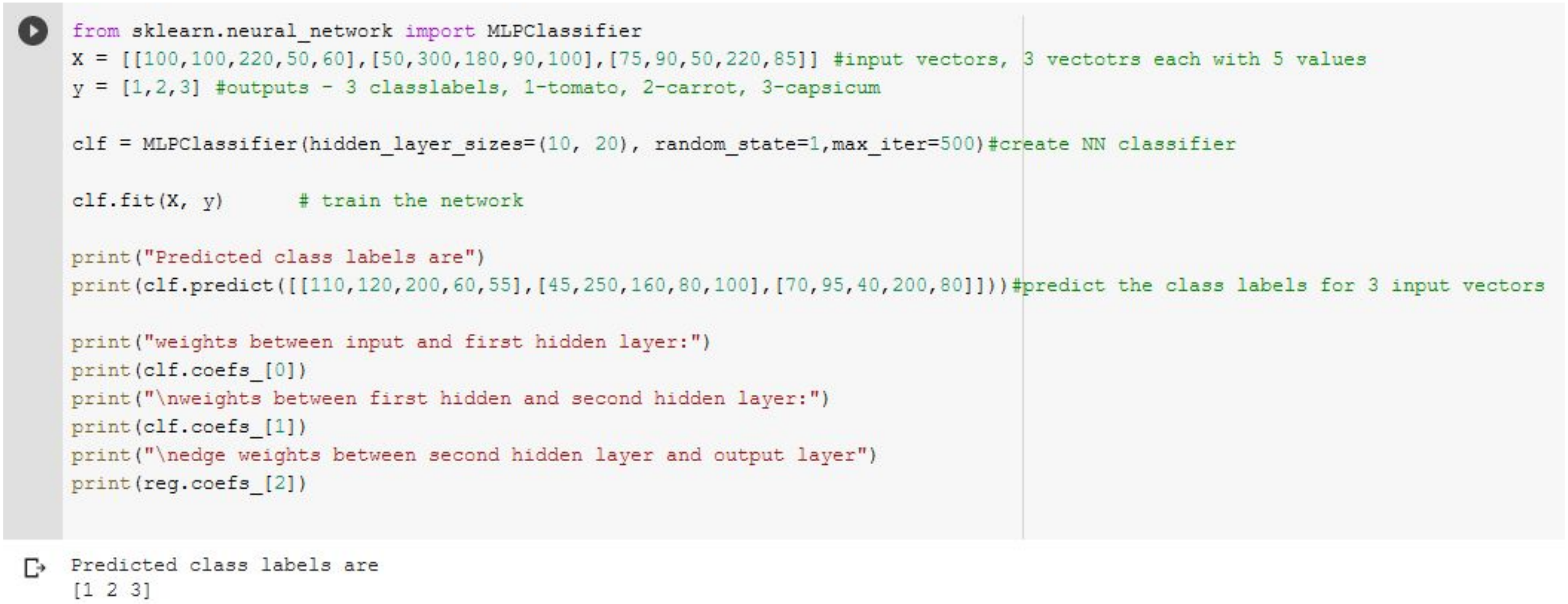
clf.fit(X, y)

3. Predict classlabel using

clf.predict(X)

↑ feature vector corresponding to unknown i/p

Neural Network Classifier



```
from sklearn.neural_network import MLPClassifier
X = [[100,100,220,50,60],[50,300,180,90,100],[75,90,50,220,85]] #input vectors, 3 vectotrs each with 5 values
y = [1,2,3] #outputs - 3 classlabels, 1-tomato, 2-carrot, 3-capsicum

clf = MLPClassifier(hidden_layer_sizes=(10, 20), random_state=1,max_iter=500)#create NN classifier

clf.fit(X, y)      # train the network

print("Predicted class labels are")
print(clf.predict([[110,120,200,60,55],[45,250,160,80,100],[70,95,40,200,80]]))#predict the class labels for 3 input vectors

print("weights between input and first hidden layer:")
print(clf.coefs_[0])
print("\nweights between first hidden and second hidden layer:")
print(clf.coefs_[1])
print("\nedge weights between second hidden layer and output layer")
print(reg.coefs_[2])
```

```
Predicted class labels are
[1 2 3]
```